



SOFTWARE, DEVICES AND METHODS FACILITATING EXECUTION OF SERVER-SIDE APPLICATIONS AT MOBILE DEVICES

COPYRIGHT NOTICE

Part of
Paper #4

5 **[0001]** A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by any one of the patent document or patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

10

CROSS-REFERENCE TO RELATED APPLICATIONS

[0002] This application claims benefits from U.S. provisional patent application no. 60/260,223 filed Jan. 9, 2001, the contents of which are hereby incorporated by reference.

15

FIELD OF THE INVENTION

[0003] The present invention relates to software, devices and methods allowing varied mobile devices to interact with server side software applications.

20

BACKGROUND OF THE INVENTION

[0004] Wireless connectivity is a feature of the modern telecommunications environment. An increasing range of people are using a wide variety of wireless data networks to access corporate data applications.

25

[0005] However, there are numerous competing mobile devices that can be used to achieve this. Each device has its own operating system and its own display characteristics. Operating systems are not mutually compatible,

nor are the display characteristics – some are color, some are black and white, some are text-only, some are pictorial.

5 **[0006]** At the same time, an increasing number of mobile device users are people without a technical background or high level of educational achievement. Such people are often intimidated by the need to run complex installation programs. Furthermore, at present, such installation programs generally depend on cable connections to a personal computer by the means of a 'cradle' or other such device.

10 **[0007]** Therefore, a mechanism whereby a mobile client for a server side application may be enabled for multiple wireless devices with minimal modification of the application at the server is required. Further, the ability to install and upgrade the application onto mobile devices wirelessly without the need for human intervention or connection to PCs, is desirable.

SUMMARY OF THE INVENTION

15

20 **[0008]** In accordance with the present invention, data from an application executing at a computing device is presented at a remote wireless device, by providing the device an application definition file, containing definitions for a user interface format for the application at the wireless device; the format of network messages for exchange of data generated by the application; and a format for storing data related to the application at the wireless device. Using these definitions, the wireless device may receive data from said application in accordance with the definition and present an interface for the application.

25 **[0009]** Preferably, the application definition file is an XML file. Similarly, application specific network messages provided to the device are also formed using XML.

[0010] In the preferred embodiment, the data from the application is presented at the mobile device by virtual machine software that uses the application definition file.

[0011] In accordance with an aspect of the present invention, a method of presenting data from an application executing at a computing device at a remote wireless device, includes: receiving at the wireless device, a representation of a text file defining: a format of a user interface for the application at the wireless device; format of network messages for exchange of data generated by the application; a format for storing data related to the application at the wireless device. Thereafter, data from the application may be received in accordance with the format of network transactions, and presented at the wireless device using the user interface.

[0012] In accordance with another aspect of the present invention, a wireless mobile device includes a processor and computer readable memory in communication with the processor, storing virtual machine software controlling operation of the device. The virtual machine software includes a parser for receiving a text file; a screen generation engine, for presenting at least one screen at the device in accordance with the text file; an event handler for processing events arising in response to interaction with the at least one screen in accordance with the text file; and object classes corresponding to actions to be taken by the in response to interaction with the at least one screen.

[0013] Other aspects and features of the present invention will become apparent to those of ordinary skill in the art, upon review of the following description of specific embodiments of the invention in conjunction with the accompanying figures.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] In figures which illustrate, by way of example, embodiments of the present invention,

[0015] FIG. 1 schematically illustrates a mobile device, exemplary of an embodiment of the present invention, including virtual machine software, further exemplary of an embodiment of the present invention;

[0016] FIG. 2 further illustrates the organization of exemplary virtual machine software at the mobile device of FIG. 1;

[0017] FIG. 3 illustrates an operating environment for the device of FIG. 1;;

[0018] FIG 4 illustrates the structure of example application definitions stored at a server of FIG. 2 used by the device of FIG. 1;;

[0019] FIG. 5 schematically illustrates the formation of application definition files at a middleware server of FIG. 2

[0020] FIG. 6 schematically illustrates the middleware server of FIG. 2, exemplary of an embodiment of the present invention, including an application definitions database, further exemplary of an embodiment of the present invention;

[0021] FIG. 7 is a flow diagram illustrating the exchange of sample messages passed between a mobile device, middleware server and application server of FIG. 2;

[0022] FIG. 8-10 illustrate steps performed at a mobile device under control of virtual machine software of FIG. 2;

[0023] FIG. 11 illustrates the format of messages exchanged in the message flow of FIG. 7;

[0024] FIG. 12 illustrates a presentation of a user interface for a sample application at a mobile device;

[0025] FIG. 13 illustrates a sample portion of an application definition file defining a user interface illustrated in FIG. 12;

[0026] FIG. 14 illustrates the format of a message formed in accordance with the sample portion of an application definition file of FIG. 13;

[0027] FIG. 15A illustrates a sample portion of an application definition file defining a local storage at a mobile device;

[0028] FIG. 15B schematically illustrates local storage in accordance with FIG. 15A; and

[0029] FIG. 15C illustrates how locally stored data is updated by a sample message in accordance with the sample portion of an application file definition of FIG. 15A;

[0030] FIGS. 16A-16JJ contain Appendix "A" detailing example XML entities understood by the virtual machine software of the mobile device of FIG. 1.

10 DETAILED DESCRIPTION

[0031] FIG. 1 illustrates a mobile device 10, exemplary of an embodiment of the present invention. Mobile device 10 may be any conventional mobile device, modified to function in manners exemplary of the present invention. As such, mobile device 10 includes a processor 12, in communication with a
15 network interface 14, storage memory 16, and a user interface 18 typically including a keypad and/or touch-screen. Network interface 14 enables device 10 to transmit and receive data over a wireless network 22. Mobile device 10 may be, for example, be a Research in Motion (RIM) two-way paging device, a WinCE based device, a PalmOS device, a WAP enabled mobile telephone,
20 or the like. Memory 16 of device 10 stores a mobile operating system such as the PalmOS, or WinCE operating system software 20. Operating system software 20 typically includes graphical user interface and network interface software having suitable application programmer interfaces ("API"s) for use by other applications executing at device 10.

25 [0032] Memory at device 10 further stores virtual machine software 24, exemplary of an embodiment of the present invention. Virtual machine software 24, when executed by mobile device 10 enables device 10 to present an interface for server side applications provided by a middleware server, described below. Specifically, virtual machine software 24 interprets a
30 text application definition file defining a user interface 18 controlling

application functionality, and the display format (including display flow) at device 10 for a particular server-side application; the format of data to be exchanged over the wireless network for the application; and the format of data to be stored locally at device 10 for the application. Virtual machine software 24 uses operating system 20 and associated APIs to interact with device 10, in accordance with the received application definition file. In this way, device 10 may present interfaces for a variety of applications, stored at a server. Moreover, multiple wireless devices each having a similar virtual machine software 24 may use a common server side application in combination with an application definition, to present a user interface and program flow specifically adapted for the device.

[0033] As such, and as will become apparent, the exemplary virtual machine software 24 is specifically adapted to work with the particular mobile device 10. Thus if device 10 is a RIM pager, virtual machine software 24 is a RIM virtual machine. Similarly, if device 10 is a PalmOS or WinCE device, virtual machine software 24 would be a PalmOS or a WinCE virtual machine. As further illustrated in FIG. 1, virtual machine software 24 is capable of accessing local storage 28 at device 10.

[0034] As detailed below, an exemplary application definition file may be formed using a markup language, like XML. In accordance with an embodiment of the present invention, defined XML entities are understood by the virtual machine software 24. Defined XML entities are detailed in Appendix "A", hereto. The defined XML entities are interpreted by the virtual machine software 24, and may be used as building blocks to present server-side applications at mobile device 10, as detailed herein.

[0035] Specifically, as illustrated in FIG. 2, virtual machine software 24 includes a conventional XML parser 61; an event handler 65; a screen generation engine 67; and object classes 69 corresponding to XML entities supported by the virtual machine software 24, and possibly contained within an application definition file. Supported XML entities are detailed in Appendix "A" hereto enclosed. A person of ordinary skill will readily appreciate that

those XML entities identified in Appendix "A" are exemplary only, and may be extended, or shortened as desired.

5 [0036] XML parser 61 may be formed in accordance with the Document Object Model, or DOM, available at <http://www.w3.org/DOM/>, the contents of which are hereby incorporated by reference. Parser 61 enables virtual machine software 24 to read an application description file. Using the parser, the virtual machine software 24 may form a binary representation of the application definition file for storage at the mobile device, thereby eliminating the need to parse text each time an application is used. Parser 61 may
10 convert each XML tag contained in the application definition file, and its associated data to tokens, for later processing. As will become apparent, this may avoid the need to repeatedly parse the text of an application description file.

15 [0037] Screen generation engine 67 displays initial and subsequent screens at the mobile device, in accordance with an application description file 28, as detailed below.

[0038] Event handler 65, of virtual machine software 24 allows device 10 under control of virtual machine software 24 to react to certain external events. Example events include user interaction with presented screens or
20 display elements, incoming messages received from a wireless network, or the like.

[0039] Object classes 69 define objects that allow device 10 to process each of the supported XML entities at the mobile device. Each of object classes 69 includes attributes used to store parameters defined by the XML
25 file, and functions allowing the XML entity to be processed at the mobile device, as detailed in Appendix "A", for each supported XML entity. So, as should be apparent, supported XML entities are extensible. Virtual machine software 24 may be expanded to support XML entities not detailed in Appendix "A". Corresponding object classes could be added to virtual
30 machine software 24.

[0040] As detailed below, upon invocation of a particular application at mobile device 10, the virtual machine software 24 presents an initial screen, based on the contents of the application definition file 28. Screen elements are created by screen generation engine 67 by creating instances of
5 corresponding object classes for defined elements, as contained within object classes 69. The object instances are created using attributes contained in the application definition file 28. Thereafter the event handler 65 of the virtual machine software 24 reacts to events for the application. Again, the event handler consults the contents of the application definition file for the
10 application in order to properly react to events. Events may be reacted to by creating instances of associated "action" objects, from object classes 69 of virtual machine software 24.

[0041] Similarly, object classes 69 of virtual machine software 24 further include object classes corresponding to data tables and network transactions
15 defined in the Table Definition and Package Definition sections of Appendix "A". At run time, instances of object classes corresponding to these classes are created and populated with parameters contained within application definition file, as required.

[0042] Using this general description, persons of ordinary skill in the art will
20 be able to form virtual machine software 24 for any particular device. Typically, virtual machine software 24 may be formed using conventional object oriented programming techniques, and existing device libraries and APIs, as to function as detailed herein. As will be appreciated, the particular format of screen generation engine 67, object classes 69 will vary depending
25 on the type of virtual machine software, its operating system and API available at the device. Once formed, a machine executable version of virtual machine software 24 may be loaded and stored at a mobile device, using conventional techniques. It can be embedded in ROM, loaded into RAM over a network, or from a computer readable medium. Although, in the preferred
30 embodiment the virtual machine software 24 is formed using object oriented structures, persons of ordinary skill will readily appreciate that other

approaches could be used to form suitable virtual machine software. For example, the object classes forming part of the virtual machine could be replaced by equivalent functions, data structures or subroutines formed using a conventional (i.e. non-object oriented) programming environment.

- 5 Operation of virtual machine software 24 under control of an application definition containing various XML definitions exemplified in Appendix "A", is further detailed below.

[0043] FIG. 3 illustrates the operating environment for a mobile device 10. Further example mobile devices 30, 32 and 34 are also illustrated in FIG. 3.

- 10 These mobile devices 30, 32 and 34 are similar to device 10 and also store and execute virtual machine software exemplary of an embodiment of the present invention.

[0044] Virtual machine software like that stored at device 10, executes on each mobile device 10, 30, 32, 34, and communicates with a middleware
15 server 44 by way of example wireless networks 36 and 38 and network gateways 40 and 42. Example gateways 40 and 42 are generally available as a service for those people wishing to have data access to wireless networks. An example network gateway is available from Broadbeam Corporation in association with the trademark SystemsGo!. Wireless networks 36 and 38 are
20 further connected to one or more computer data networks, such as the Internet and/or private data networks by way of gateway 40 or 42. As will be appreciated, the invention may work with many types of wireless networks. Middleware server 44 is in turn in communication with a data network, that is in communication with wireless network 36 and 38. The communication used
25 for such communication is via TCP/IP over an HTTP transport. As could be appreciated, other network protocols such as X.25 or SNA could equally be used for this purpose.

- [0045]** Devices 10, 30, 32, and 34 communicate with middleware server 44 in two ways. First, virtual machine software 24 at each device may query
30 middleware server 44 for a list of applications that a user of an associated mobile device 10,30,32 or 34 can make use of. If a user decides to use a

particular application, device 10, 30, 32 or 34 can download a text description, in the form of an application definition file, for the application from the middleware server 44 over its wireless interface. As noted, the text description is preferably formatted using XML. Second, virtual machine software 24 may
5 send, receive, present, and locally store data related to the execution of applications, or its own internal operations. The format of exchanged data for each application is defined by an associated application description file. Again, the exchanged data is formatted using XML, in accordance with the application description file.

10 **[0046]** Middleware server 44, in turn, stores text application description files for those applications that have been enabled to work with the various devices 10, 30, 32, and 34 using virtual machine software 24 in a pre-defined format understood by virtual machine software 24. Software providing the functions of the middleware server 44, in the exemplary embodiment is written
15 in Delphi, using an SQL Server database.

[0047] As noted, text files defining application definitions and data may be formatted in XML. For example XML version 1.0, detailed in the XML version 1.0 specification second edition, dated 6th October 2000 and available at the internet address "<http://www.w3.org/TR/2000/REC-xml-20001006>", the
20 contents of which are hereby incorporated herein by reference, may be used. However, as will be appreciated by those of ordinary skill in the art, the functionality of storing XML description files is not dependent on the use of any given programming language or database system.

[0048] Each application definition file is formatted according to defined
25 rules and uses pre-determined XML markup tags, known by both virtual machine software 24, and complementary middleware server software 68. Tags define XML entities used as building blocks to present an application at a mobile device. Knowledge of these rules, and an understanding of how each tag and section of text should be interpreted, allows virtual machine
30 software 24 to process an XML application definition and thereafter execute an application, as described below. Virtual machine software 24 effectively

acts as an interpreter for a given application definition file.

[0049] FIG. 4 illustrates an example format for an XML application definition file 28. As illustrated, the example application definition file 28 for a given device and application includes three components: a user interface definition section 48, specific to the user interface for the device 10, and defining the format of screen or screens for the application and how the user interacts with them; a network transactions definition section 50 defining the format of data to be exchanged with the application; and a local data definition section 52 defining the format of data to be stored locally on the mobile device by the application.

[0050] Defined XML markup tags correspond to XML entities supported at a device, and are used to create an application definition file 28. The defined tags may broadly be classified into three categories, corresponding to the three sections 48, 50 and 52 of an application definition file 28.

[0051] Example XML tags and their corresponding significance are detailed in Appendix "A". As noted above, virtual machine software 24 at a mobile device includes object classes corresponding to each of the XML tags. At run time, instances of the objects are created as required.

[0052] Broadly, the following example XML tags may be used to define the user interface definition:

[0053] SCREEN – this defines a screen. A SCREEN tag pair contains the definitions of the user interface elements (buttons, radio buttons, and the like) and the events associated with the screen and its elements

[0054] BUTTON – this tag defines a button and its associated attributes

[0055] LIST – this tag defines a list box

[0056] CHOICEBOX – this tag defines a choice item, that allows selection of a value from predefined list

[0057] MENU – the application developer will use this tag to define a menu for a given screen

[0058] EDITBOX – this tag defines an edit box

[0059] TEXT ITEM – this tag describes a text label that is displayed

5 **[0060]** CHECKBOX – this tag describes a checkbox

[0061] HELP – this tag can define a help topic that is used by another element on the screen

[0062] IMAGE – this tag describes an image that appears on those displays that support images

10 **[0063]** ICON – this tag describes an icon

[0064] EVENT – this defines an event to be processed by the virtual machine software. Events can be defined against the application as a whole, individual screens or individual items on a given screen. Sample events would be receipt of data over the wireless interface, or a edit of text in an edit box

15 **[0065]** ACTION – this describes a particular action that might be associated with an event handler. Sample actions would be navigating to a new window or displaying a message box.

[0066] The second category of example XML tags describes the network transaction section 50 of application definition 28. These may include the
20 following example XML tags;

[0067] TABLEUPDATE – using this tag, the application developer can define an update that is performed to a table in the device's local storage. Attributes allow the update to be performed against multiple rows in a given table at once;

25 **[0068]** PACKAGEFIELD – this tag is used to define a field in a data package that passes over the wireless interface

[0069] The third category of XML tags used to describe an application are those used to define a logical database that may be stored at the mobile device. The tags available that may be used in this section are:

5 **[0070]** TABLE – this tag, and its attributes, define a table. Contained within a pair of TABLE tags are definitions of the fields contained in that table. The attributes of a table control such standard relational database functions as the primary key for the table.

10 **[0071]** FIELD – this tag describes a field and its attributes. Attributes of a field are those found in a standard relational database system, such as the data type, whether the field relates to one in a different table, the need to index the field, and so on.

15 **[0072]** As well as these XML tags, virtual machine software 24 may, from time to time, need to perform certain administrative functions on behalf of a user. In order to do this, one of object classes 69 has its own repertoire of tags to communicate its needs to the middleware server 44. Such tags differ from the previous three groupings in that they do not form part of an application definition file, but are solely used for administrative communications between the virtual machine software 24 and the middleware server 44. Data packages using these tags are composed and sent due to user interactions with the virtual machine's configuration screens. The tags used for this include,

[0073] REG – this allows the application to register and deregister a user for use with the middleware server

25 **[0074]** FINDAPPS – by using this operation, users can interrogate the server for the list of applications that are available to them

[0075] APP REG – using this operation, the end-user can register (or deregister) for an application and have the application interface downloaded automatically to their device (or remove the interface description from the device's local storage).

[0076] SETACTIVE – If the user's preferred device is malfunctioning, or out of power or coverage, they will need a mechanism to tell the Server to attempt delivery to a different device. The SETACTIVE command allows the user to set the device that they are currently using as their active one

5 **[0077]** FIG. 5 illustrates the organization of application definitions at middleware server 44 and how middleware server 44 may form an application definition file 28 (FIG. 4) for a given device 10, 30, 32 or 34. In the illustration of FIG. 5, only two mobile devices 10 and 30 are considered. Typically, since network transactions and local data are the same across devices, the only
10 piece of the application definition that varies for different devices is the user interface definition.

[0078] So, middleware server 44 stores a master definition 58 for a given server side application. This master definition 58 contains example user interface descriptions 48, 54, 56 for each possible mobile device 10, 30, 32;
15 descriptions of the network transactions 50 that are possible and data descriptions 52 of the data to be stored locally on the mobile device. Preferably, the network transactions 50 and data descriptions 52 will be the same for all mobile devices 10, 30 and 32.

[0079] For device 10, middleware server 44 composes an application
20 definition file 28 by querying the device type and adding an appropriate user interface description 48 for device 10 to the definitions for the network transactions 50 and the data 52. For device 30, middleware server 44 composes the application definition by adding the user interface description 54 for device 10 to the definitions for the network transactions 50 and data 52.

25 **[0080]** The master definition 58 for a given application is created away from the middleware server and loaded onto the middleware server by administrative staff charged with its operation. Master definition files could be created either by use of a simple text editor, or by a graphical file generation tool. Such a tool might generate part or all of the file, using knowledge of the
30 XML formatting rules, based on the user's interaction with screen painters,

graphical data definition tools and the like.

5 [0081] FIG. 6 illustrates the organization of middleware server 44 and associated master definitions. Middleware server 44 may be any conventional application server, modified to function in manners exemplary of the present invention. As such, middleware server 44 includes a processor 60, in communication with a network interface 66 and storage memory 64. Middleware server 44 may be, for example, be a Windows NT server, a Sun Solaris server, or the like. Memory of middleware server 44 stores an operating system such as Windows NT, or Solaris operating system software
10 62.

[0082] Network interface 66 enables middleware server 44 to transmit and receive data over a data network 63. Transmissions are used to communicate with both the virtual machine software 24 (via the wireless networks 36, 38 and wireless gateways 40,42) and one or more application
15 servers, such as application server 70, that are the end recipients of data sent from the mobile client applications and the generators of data that is sent to the mobile client applications.

[0083] Memory at middleware server 44 further stores software 68, exemplary of an embodiment of the present invention. Middleware server
20 software 68, when executed by middleware server 44 enables the middleware server to understand and compose XML data packages that are sent and received by the middleware server. These packages may be exchanged between middleware server 44 and the virtual machine software 24, or between the middleware server 44 and the application server 70.

25 [0084] As described above, communication between the application server 70 and the middleware server 44 uses HTTP running on top of a standard TCP/IP stack. An HTTP connection between a running application at the application server 70 and the middleware server 44 is established in response to the application at a mobile device presenting the application. The server
30 side application provides output to middleware server 44 over this connection.

The server side application data is formatted into appropriate XML data packages understood by the virtual machine software 24 at a mobile device by the server side application.

5 **[0085]** That is, a server side application (or an interface portion of the application) formats application output into XML in a manner consistent with the format defined by the application definition file for the application. Alternatively, an interface component separate from the application could easily be formed with an understanding of the format and output for a particular application. That is, with a knowledge of the format of data provided
10 and expected by an application at application server 70, an interface component could be produced using techniques readily understood by those of ordinary skill. The interface portion could translate application output to XML, as expected by middleware server 44. Similarly, the interface portion may translate XML input from a mobile device into a format understood by the
15 server side application.

20 **[0086]** The particular identity of the mobile device on which the application is to be presented may be identified by a suitable identifier, in the form of a header contained in the server side application output. This header may be used by middleware server 44 to forward the data to the appropriate mobile device. Alternatively, the identity of the connection could be used to forward the data to the appropriate mobile device.

25 **[0087]** FIG. 7 illustrates a flow diagram detailing data (application data or application definition files 28) flow between mobile device 10 and middleware server 44, in manners exemplary of an embodiment of the present invention.

30 **[0088]** For data requested from middleware server 44, device 10, under software control by virtual machine software 24 makes requests to middleware server 44 (also illustrated in FIG. 2), which passes over the wireless network 36 through network gateway 40. Network gateway 40 passes the request to the middleware server 44. Middleware server 44 responds by executing a database query on its database 46 that finds which

applications are available to the user and the user's mobile device. For data passed from middleware server 44 to device 10, data is routed through network gateway 40. Network gateway 40 forwards the information to the user's mobile device over the wireless network 36.

5 **[0089]** FIG. 7 when considered with FIG. 3 illustrates a sequence of communications between device 10, and middleware server 44 that may occur when the user of a mobile device wishes to download an application definition file 28 for a server side application.

10 **[0090]** So, initially, device 10 interrogates server 44 to determine which applications are available for the particular mobile device being used. This may be accomplished by the user instructing the virtual machine software 24 at device 10 to interrogate the server 44. Responsive to these instructions the virtual machine software 24 sends an XML message to the server requesting the list of applications (data flow 72); as illustrated in FIG. 7 the XML
15 message may contain the <FINDAPPS> tag, signifying to the middleware server 44, its desire for a list available application. In response, middleware server 44 makes a query to database 46. Database 46, responsive to this query, returns a list of applications that are available to the user and the mobile device. The list is typically based, at least in part, on the type of
20 mobile device making the request, and the applications known to middleware server 44. Middleware server 44 converts this list to an XML message and sends to the virtual machine (data flow 74). Again, a suitable XML tag identifies the message as containing the list of available applications.

25 **[0091]** In response, a user at device 10 may choose to register for an available server side application. When a user chooses to register for an application, virtual machine software 24 at device 10 composes and sends an XML registration request for a selected application (data flow 76) to middleware server 44. As illustrated in FIG. 8, an XML message containing a < REG> tag is sent to middleware server 44. The name of the application is
30 specified in the message. The middleware server 44, in response, queries its database for the user interface definition for the selected application for the

user's mobile device. Thereafter, the middleware server creates the application definition file, as detailed with reference to FIG.5. Then, middleware server 44 sends to the mobile device (data flow 78) the created application definition file 28.

5 **[0092]** The user is then able to use the functionality defined by the interface description to send and receive data.

[0093] At this time, parser 61 of virtual machine software 24 may parse the XML text of the application definition file to form a tokenized version of the file. That is, each XML tag may be converted a defined token for compact storage, and to minimize repeated parsing of the XML text file. The tokenized version
10 of the application definition file may be stored for immediate or later use by device 10.

[0094] Thereafter, upon invocation of a particular application for which the device 10 has registered, the screen generation engine 67 of the virtual
15 machine software 24 at the device causes the virtual device to locate the definition of an initial screen for that application. The initial screen is identified within the application definition file 28 for that application using a <SCREEN> tag, and an associated attribute of <First screen = "yes">.

[0095] Steps performed by virtual machine software 24 in processing this
20 screen (and any screen) are illustrated in FIG.8. As illustrated, screen generation engine 67, generates an instance of an object class, defining a screen by parsing the section of the XML application definition file corresponding to the <SCREEN> tag in step S802. Supported screen elements may be buttons, edit boxes, menus, list boxes, and choice items, as
25 identified in sections 5.3, 5.4, and 5.5 of Appendix "A". Other screen elements, such as images and checkboxes, as detailed in Appendix "A" may also be supported. For clarity of illustration, their processing by screen generation engine 67 however, is not detailed. Each supported tag under the SCREEN definition section, in turn causes creation of instances of object
30 classes within the virtual machine software 24. Typically, instances of objects

corresponding to the tags, used for creation of a screen, result in presentation of data at mobile device 10. As well the creation of such objects may give rise to events (e.g. user interaction) and actions to be processed at device 10.

5 **[0096]** Each element definition causes virtual machine software 24 to use the operating system of the mobile device to create corresponding display element of a graphical user interface as more particularly illustrated in FIG. 9. Specifically, for each element, the associated XML definition is read in step S806, S816, S826, S836, and S846, and a corresponding instance of a screen object defined as part of the virtual machine software 24 is created by
10 the virtual machine software 24 in steps S808, S818, S828, S838 and S848, in accordance with steps S902 and onward illustrated in FIG. 9. Each interface object instance is created in step S902. Each instance takes as attribute values defined by the XML text associated with the element. A method of the virtual object is further called in step S904, and causes a
15 corresponding device operating system object to be created. Those attributes defined in the XML text file, stored within the virtual machine object instance are applied to the corresponding instance of a display object created using the device operating system in steps S908S-S914. These steps are repeated for all attributes of the virtual machine object instance. For any element allowing
20 user interaction, giving rise to an operating system event, the event handler 65 of virtual machine software 24 is registered to process operating system events, as detailed below.

25 **[0097]** Additionally, for each event (as identified by an <EVENT> tag) and action (as identified by an <ACTION> tag) associated with each XML element, virtual machine software 24 creates an instance of a corresponding event and action object forming part of virtual machine software 24. Virtual machine software 24 further maintains a list identifying each instance of each event and action object, and an associated identifier of an event in steps S916 to S928.

30 **[0098]** Steps S902-S930 are repeated for each element of the screen in steps S808, S818, S828, S838 and S848 as illustrated in FIG. 8. All elements

between the <SCREEN> definition tags are so processed. After the entire screen has been so created in memory, it is displayed in step S854, using conventional techniques.

5 **[0099]** As will be appreciated, objects specific to the type of device executing the virtual machine software 24. Functions initiated as a result of the XML description may require event handling. This event handling is processed by event handler 65 of virtual machine software 24 in accordance with the application definition file 28. Similarly, receipt of data from a mobile network will give rise to events. Event handler 65, associated with a particular application presented at the device similarly processes incoming messages for that particular application. In response to the events, virtual machine software 24 creates instance of software objects, and calls functions of those object instances, as required by the definitions contained within the XML definitions contained within the application definition file 28, giving rise to the event.

10

15

20 **[00100]** As noted, the virtual machine software 24 includes object classes, allowing the virtual machine to create object instances corresponding to an <EVENT> tag. The event object classes includes methods specific to the mobile device that allow the device to process each of the defined XML descriptions contained within the application definition file, and also to process program/event flow resulting from the processing of each XML description.

25 **[00101]** Events may be handled by virtual machine software 24 as illustrated in FIG. 10. Specifically, as device handler 65 has been registered with the operating system for created objects, upon occurrence of an event, steps S1002 and onward are performed in response to the operating system detecting an event.

30 **[00102]** An identifier of the event is passed to event handler 65 in step S1002. In steps S1004-S1008, this identifier is compared to the known list of events, created as a result of steps S916-S930. For an identified event, actions associated with that event are processed in step S1008-S1014.

[00103] That is, virtual machine software 24 performs the action defined as a result of the <ACTION> tag associated with the <EVENT> tag corresponding to the event giving rise to processing by the event handler 65. The <ACTION> may cause creation of a new screen, as defined by a screen tag, a network transmission, a local storage, or the like.

[00104] New screens, in turn, are created by invocation of the screen generation engine 61, as detailed in FIGS. 8 and 9. In this manner the navigation through the screens of the application is accomplished according to the definition embodied in the XML application description.

[00105] Similarly, when the user wishes to communicate with the middleware server, or store data locally, event handler 65 creates instances of corresponding object classes within the object classes 69 of virtual machine software 24 and calls their methods to store or transmit the data using the local device operating system. The format of data is defined by the device local definition section 52; the format of network packages is defined in the network transaction package definition section 50.

[00106] For example, data that is to be sent to the wireless network is assembled into the correct XML packages using methods within an XML builder object, formed as a result of creating an instance of a corresponding object class within object classes 69 of virtual machine software 24. Methods of the XML builder object create a full XML package before passing the completed XML package to another message server object. The message server object uses the device's network APIs to transmits the assembled data package across the wireless network.

[00107] Received XML data packages from network 63 (FIG. 2) give rise to events processed by event handler 65. Processing of the receipt of data packages is not specifically illustrated in FIG.9. However, the receipt of data triggers a "data" event of the mobile device's operating system. This data event is passed to the virtual machine, and event handler 65 inspects the package received. As long as the data received is a valid XML data package

as contained within the application definition, the virtual machine inspects the list of recognised XML entities.

[00108] So, for example, a user could send a login request 80 by interacting with an initial login screen, defined in the application definition file for the application. This would be passed by the middleware server 44 to the backend application server 70. The backend application server according to the logic embedded within its application, would return a response, which the middleware server 44 would pass to the virtual machine software 24. Other applications, running on the same or other application servers might involve different interactions, the nature of such interactions being solely dependent on the functionality and logic embedded within the application server 70, and remaining independent of the middleware server 44.

[00109] FIG. 11 illustrates sample XML messages passed as the result of message flows illustrated in FIG. 6. For each message, the header portion, between the <HEAD>...</HEAD> tags contains a timestamp and the identifier of the sending device.

[00110] Example message 72 is sent by the mobile device to request the list of applications that the server has available to that user on that device. It specifies the type of device by a text ID contained between the <PLATFORM>...</PLATFORM> tags. Example message 74 is sent in response to message 70 by middleware server 44 to the mobile device 10. It contains a set of <APP>...</APP> tag pairs, each of which identifying a single application that is available to the user at device 10. Example message 76 is sent from the mobile device 10 to middleware server 44 to register for a single server side application. The tags specify information about the user. Message 78 is sent by the middleware server 44 to the mobile device in response to a request to register device 10 for an application. The pair of tags <VALUE>...</VALUE> gives a code indicating success or failure. In the sample message shown, a success is shown, and is followed by the interface description for the application, contained between the <INTERFACE>...</INTERFACE> tags. This interface description may then

be stored locally within memory 16 of device 10.

[00111] As noted, when a user starts an application that has been downloaded in the manner described above, the virtual machine software 24 reads the interface description that was downloaded for that device 10, and the virtual machine software 24 identifies the screen that should be displayed on startup, and displays its elements as detailed in relation to FIGS. 9 and 10. The user may then use the functionality defined by the user interface definition section 48 of the application definition 28 to send and receive data from a server side application.

[00112] For the purposes of illustration, FIGS. 12 and 13 illustrate the presentation of a user interface for a sample screen on a Windows CE Portable Digital Assistant. As illustrated in FIG. 13, a portion of an application definition file 28 defines a screen with the name 'New Msg'. This interface description may be contained within the user interface definition section 48 of an application definition file 28 associated with the application. The screen has a single button identified by the <'BTN NAME'="OK", CAPTION="Send" INDEX="0"> tag, and identified as item D in FIG. 12. This button gives rise to a single event, (identified by the <EVENTS NUM="1" tag) giving rise to a single associated action (defined by the tag <ACTION TYPE = "ARML">). This action results in the generation of a network package (defined by the tag <PKG TYPE="ME">), having an associated data format as defined between the corresponding tags. Additionally, the screen defines three editboxes, as defined after the <EDITBOXESNUM=3> tag, and identified as items A, B, and C.

[00113] Upon invocation of the application at the local device, screen generation engine 67 of virtual machine software 24 at the device process the screen definition, as detailed with reference to FIGS. 8 and 9. That is, for each tag D, the screen generation engine 67 creates a button object instance, in accordance with steps S804-S812. Similarly for each tag A, B and C within the application definition file, virtual machine software 24 at the device creates instances of edit box objects (i.e. steps S834 – S842 (FIGS. 8 and 9)). The

data contained within the object instances reflects the attributes of the relevant button and edit box tags, contained in the application definition 28 for the application.

[00114] The resulting screen at the mobile device is illustrated in FIG.

- 5 12. Each of the screen items is identified with reference to the XML segment within XML portion 92 giving rise to the screen element. The user interface depicts a screen called 'NewMsg', which uses the interface items detailed in FIG 8., but which adds the ability to compose and send data. This screen has three edit boxes, named 'To', 'Subject' and 'Body' as displayed in FIG 8
10 (84,86,88); these are represented by the XML tags A,B and C. The screen also incorporates a button, named 'OK', also as displayed in FIG 12 (90), which is represented by the XML tag D.

[00115] Call-backs associated with the presented button cause graphical user interface application software/operating system at the mobile device to
15 return control to the event handler 65 of virtual machine software 24 at the device. Thus, as the user interacts with the application, the user may input data within the presented screen using the mobile device API. Once data is to be exchanged with middleware server 44, the user may press the OK button, thereby invoking an event, initially handled by the operating system of
20 the mobile device. However, during the creation of button D, in steps S804-S810 any call-back associated with the button was registered to be handled by event handler 65 of virtual machine software 24. Upon completion, virtual machine software 24 receives data corresponding to the user's interaction with the presented user interface and packages this data into XML messages
25 using corresponding objects, populated according to the rules within the application definition file.

[00116] Event handler 65, in turn processes the event caused by interaction of the button in accordance with the <EVENT> tag and corresponding <ACTION> tag associated with the button D. The events, and
30 associated actions are listed as data items associated with the relevant user interface item, as result of the EVENT and ACTION tags existing within the

definitions of the relevant user interface item, within the application definition file. This <ACTION> tag causes the virtual machine software 24 to create an instance of an object that sends an XML package to the middleware server in accordance with the format defined between the <ACTION> tag. That is, a
5 "template" (defined after the <PKG TYPE="ME"> tag) for the XML package to be sent is defined against the EVENT handler for a given user interface item. This template specifies the format of the package to be sent, but will include certain variable fields. These are pieces of data in the formatted XML package that will vary according to the values contained in data entry fields on
10 the current and previous screens. The definition of the template specifies which data entry field should be interrogated to populate a given entry within a data package that is to be sent.

[00117] This template fills some of its fields dynamically from data inserted by a user into edit boxes that were presented on the mobile device's
15 screen. The template has within it certain placeholders delimited by square brackets ([,]). These placeholders specify a data source from which that section of the template should be filled. A data source might be a user interface field on the current screen, a user interface field on the previous screen, or a database table. Virtual machine software 24, reading the data
20 source name, searches for the field corresponding to that data source and replaces the placeholder with actual data contained within the data source. For example, the SUBJECT attribute of the MAIL tag in XML portion 92 is read from the edit box named 'Subject' on the screen named 'NewMsg' This process is repeated for each such placeholder, until the virtual machine,
25 reading through the template has replaced all placeholders in the template. At this point the template has been converted into a well-formed XML message 94.

[00118] A resulting XML message 94 containing data formed as a result of input provided to the fields of the "NewMsg" screen is illustrated in FIG. 14.
30 This exemplary XML message 94 that is created by pressing the button 90 in XML message portion 92. In this case, the editbox 86 named 'Subject'

contains the text "Hello Back"; the editbox 84 named 'To' contains the text "steveh@nextair.com"; and the editbox 88 named 'Body' contains the text "I am responding to your message".

5 **[00119]** The virtual machine software 24 using the template inspects these three fields, and places the text contained within each edit box in the appropriate position in the template. For example, the placeholder [NewMsg.Subject] is replaced by "Hello Back". The virtual machine software 24, inspecting the template contained in the XML message portion 92 and populating the variable fields, creates the sample XML message 94 by
10 invoking the functionality embedded within an XML builder software object. Once the XML message 94 has been assembled in this fashion, the relevant method of the message server object is then invoked to transmit the XML message 94 in a data package across the network.

15 **[00120]** Similarly, when data is received, the event handler 65 of the virtual machine software 24 is notified. In response, the event handler examines the data package that it has received using the parser 61 to build a list of name value pairs containing the data received. Thereafter, methods within an object class for processing incoming packets are invoked to allow virtual machine software 24 to inspect the application definition for the
20 application to identify the fields in the database and user interface screens that need to be updated with the new data. Where screens are updated, this is done according to the procedures normal to that device.

25 **[00121]** Handling of incoming packages is defined in the application definition file 28 at the time the application description file was downloaded. That is, for each of the possible packages that can be received, application description file 28 includes definitions of database tables and screen items that should be updated, as well as which section of the package updates which database or screen field. When a package is received, event handler 65 of virtual machine software 24 uses rules based on the application
30 description file 28 to identify which database and screen fields need to be updated.

[00122] FIGS. 15A-15C similarly illustrates how local storage on the device, and the messages that update it, are defined in the application definition file 28. XML portion 96 forming part of the device local definition section 52 of an application definition defines an example format of local storage for the email application described in FIGS. 12 and 13. Two example tables, labeled E and F are defined in the local storage for the application. One table (E) stores details of sent emails. A second table (F) stores the recipients of sent emails. The first table E, "SentItems", has four fields; the second table F, "Recipients" has three fields. This is illustrated in graphical form below the XML fragment.

[00123] FIGS. 15A and 15B further illustrates the use of local storage to store to data packages that are sent and received. Specifically, as illustrated in FIG. 15A the table given in FIG 15A may store an email contained in the example message 94, shown in FIG. 14. So application definition file 28 for this application would contain, along with XML message portions 92 and XML portion 96, the XML fragment 102. XML fragment 102 defines how the data packages composed by the XML message portion 92 (an example of which was illustrated in FIG. 13), updates the tables defined by the XML portion 96.

[00124] XML fragment 102 includes two sections 104 and 106. First section 104 defines how the fields of the data package would update the "SentItems" table E. An example line 108 describes how the 'MSGID' field in the data package would update the 'LNGMESSAGEID' field in the table E. Similarly, the second section 106 describes how the fields of the data package would update the "Recipients" table.

[00125] Attributes of the illustrated <AXDATAPACKET> tag instruct the virtual machine software 24 as to whether a given data package should update tables in local storage. These rules are applied whenever that package is sent or received.

[00126] As can be seen from the preceding description and example, such an approach has significant advantages over the traditional method of

deploying applications onto mobile devices. First, the definition of an application's functionality is separated from the details associated with implementing such functionality, allowing the implementers of a mobile application to concentrate on the functionality and ignore implementation details. Second, application definitions can be downloaded wirelessly, wherever the device happens to be at the time. This greatly improves the usefulness of the mobile device, by removing reliance on returning the device to a cradle and running a complex installation program. Thirdly, the use of application definition files allows flexible definitions for numerous applications. Server-side application may be easily ported for a number of devices.

[00127] It will be further understood that the invention is not limited to the embodiments described herein which are merely illustrative of a preferred embodiment of carrying out the invention, and which is susceptible to modification of form, arrangement of parts, steps, details and order of operation. The invention, rather, is intended to encompass all such modification within its scope, as defined by the claims.